MontanaTech
THE UNIVERSITY OF MONTANA

CSCI 136: Fundamentals of Computer Science II

Socket Programming

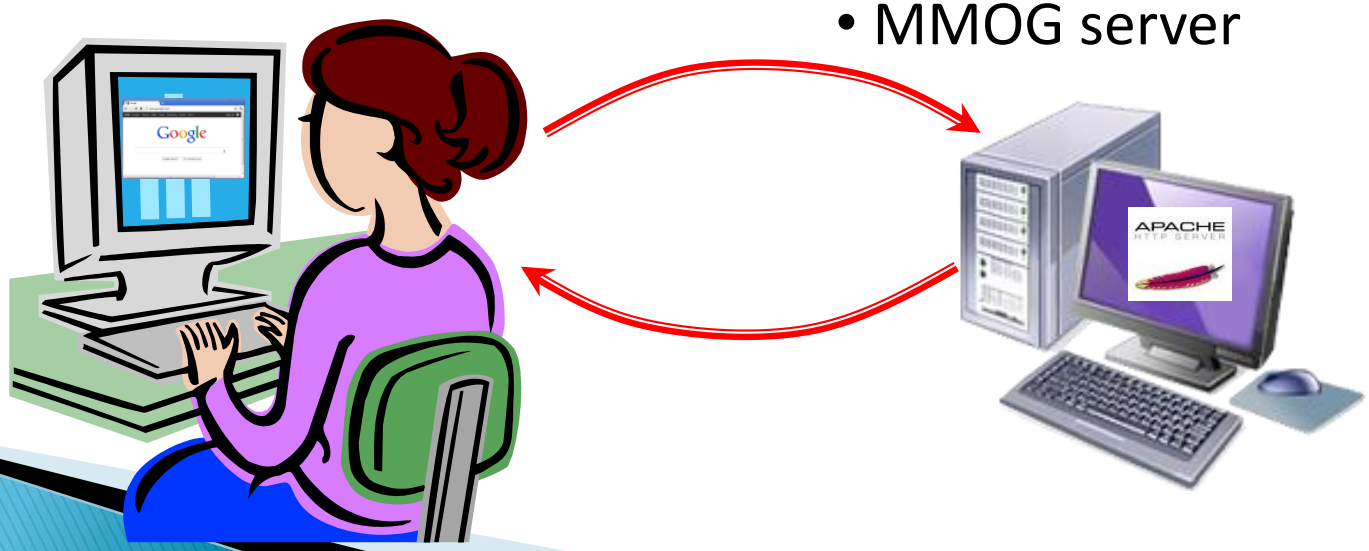
Michele Van Dyne
MUS 204B
mvandyne@mtech.edu
https://katie.mtech.edu/classes/csci136

Outline

- Networking basics
  - Difference between: clients and servers
  - Addressing
    - IP addresses, hostnames, DNS
    - Private addresses, localhost
  - Port numbers
- Socket communication
  - Java client: reading/writing text
  - Java server: accepting clients, reading/writing text

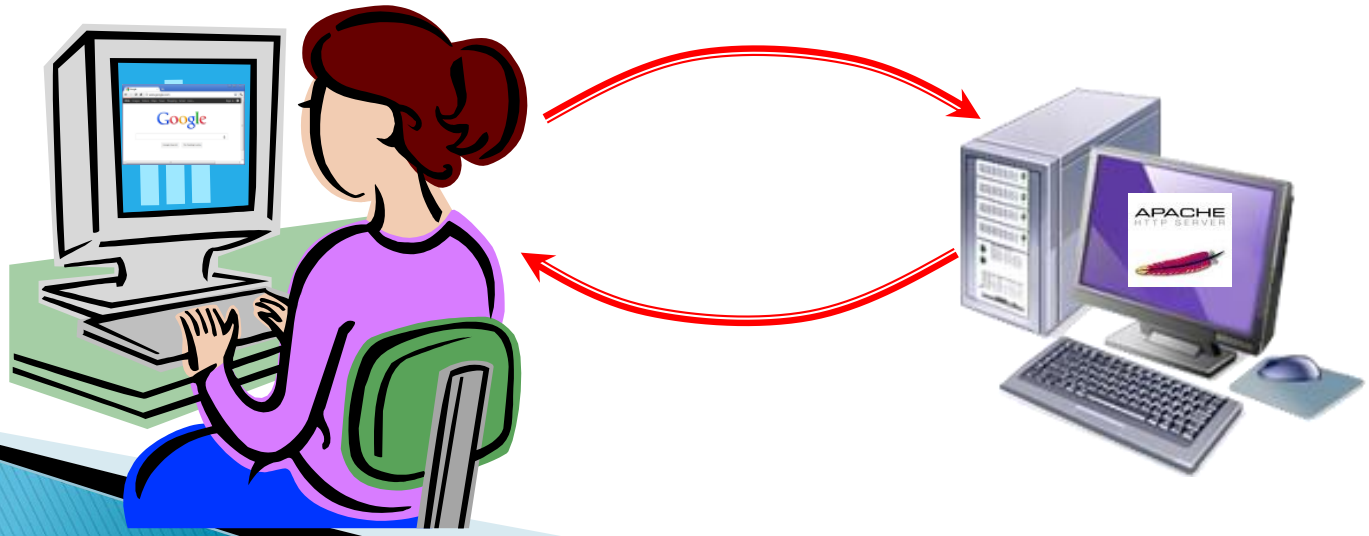
Clients and servers

- Client program
  - Requests a service
    - Web browser
    - Streaming audio player
    - Twitter client
    - MMOG client
- Server program
  - Provides a service
    - Web server
    - Streaming audio from radio station
    - Server at Twitter
    - MMOG server



Clients and servers

- Client program
  - "sometimes on"
  - Doesn't talk to other clients
  - Needs to know server's address
- Server program
  - "always on"
  - Handles requests from many clients
  - Needs fixed address



Communication components

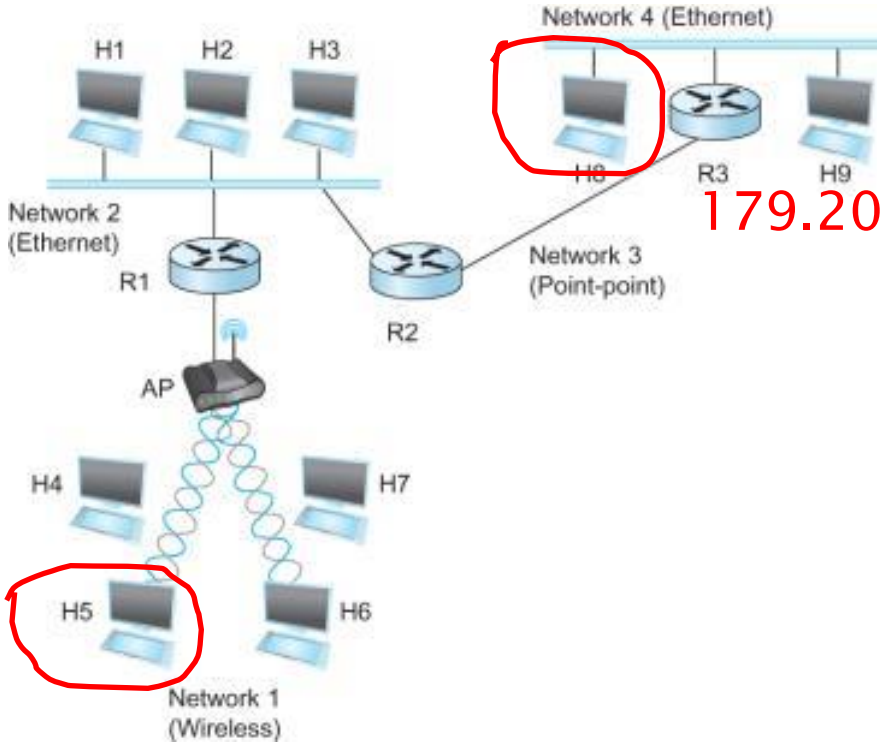
- Network
  - Transports data from source to destination host
  - Uses destination IP address
- Operating system
  - Data is forwarded to a "silo" based on port #
    - e.g. Port 80 requests routed to the web server program
- Application
  - Actually reads and writes to socket
  - Implements application-specific "magic"
    - e.g. Implementing a mail reading/writing protocol
    - e.g. Implementing a file retrieval (FTP) protocol
    - e.g. Implementing a particular online game

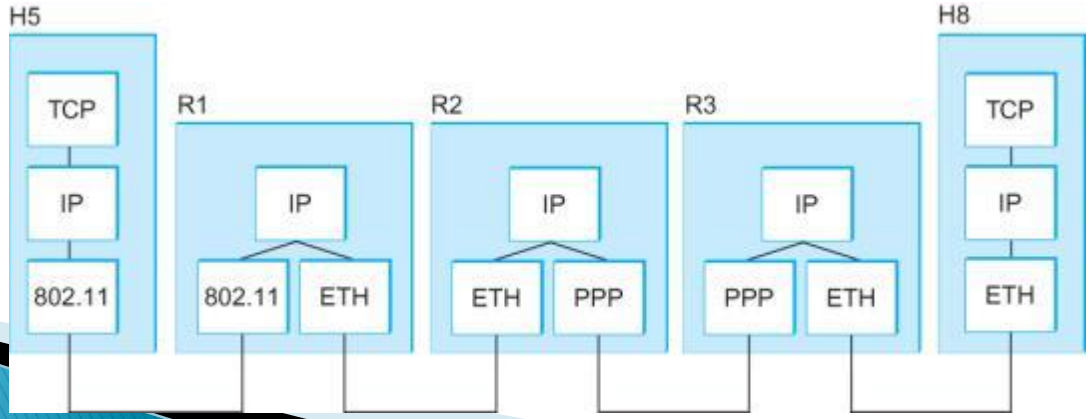
Naming computers

- Goal: Establish communication between A and B
  - How do computer A and B refer to each other?
  - The network needs an addressing system

- IP (Internet Protocol) address
  - IPv4 address
    - 32 bits ~ 4 billion hosts
    - Usually expressed as four numbers 0–255 (8 bits)
    - e.g. 173.194.79.106
  - IP address uniquely identifies a network endpoint
  - Devices inside network (e.g. switches, routers) use a packet's IP address to get it to its destination

Communication from H5 to H8





DNS – Domain Name System

- Problem 1: Humans can't remember all the numbers in an IP address
- Domain Name System (DNS)
  - Converts readable name to numeric IP address
    - e.g. www.google.com -> 173.194.79.106

http://xkcd.com/302/

Private IP addresses

- Private IP addresses
  - Allow construction of a private network
    - Route data between endpoints on the private network
    - Addresses aren't valid outside network
    - 192.168.x.x, 10.x.x.x, 172.16/31.x.x
  - Typically what you'll have:
    - On home network
    - On campus network (wired/wireless)
  - 127.0.0.1 (localhost)

http://xkcd.com/742/

Port numbers

- Problem 3: Many apps on same computer want to talk at same time
  - Chrome process:
    - Browser tab 1 wants: `http://google.com`
    - Browser tab 2 wants: `http://google.com/gmail`
    - Browser tab 3 wants: `http://facebook.com`
  - Thunderbird process:
    - Email client wants IMAP4 to `techmail.mtech.edu`
- Solution: Use IP address + port number
  - A 16-bit number: `0 - 65535`
    - Port number determines app message is routed to
    - Just a "virtual" port, only exists in the OS

Port numbers

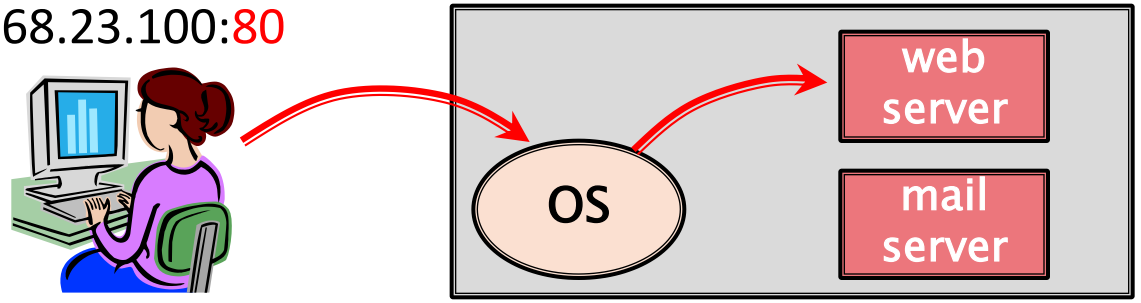
- Popular applications have known ports
  - Ports 0 – 1023: reserved for well-known services
    - Only administrators can start servers on these ports
  - Ports 1024 – 65535: available to any user-level application

Port	Service
21	File transfer protocol (FTP)
22	Secure shell (SSH)
23	Telnet
25	Simple mail transfer protocol (SMTP)
53	Domain name system (DNS)
80	Hypertext transfer protocol (HTTP)
110	Post office protocol (POP)
143	Internet message access protocol (IMAP)
443	HTTP secure (HTTPS)

Use of port number

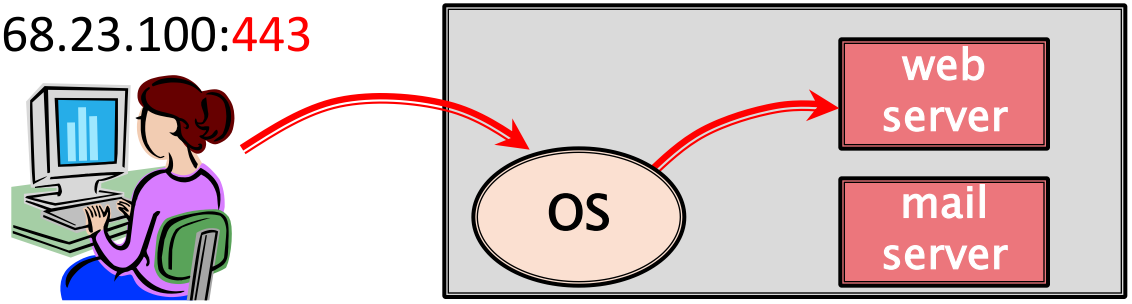
Requesting a non-secure web page

192.168.23.100:80



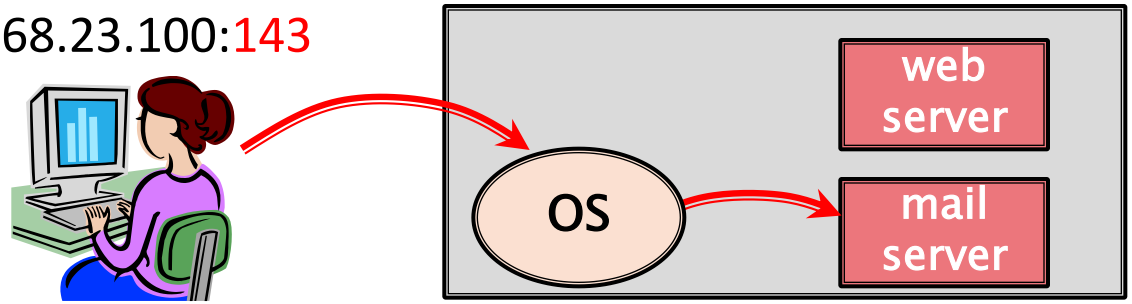
Requesting a secure web page

192.168.23.100:443



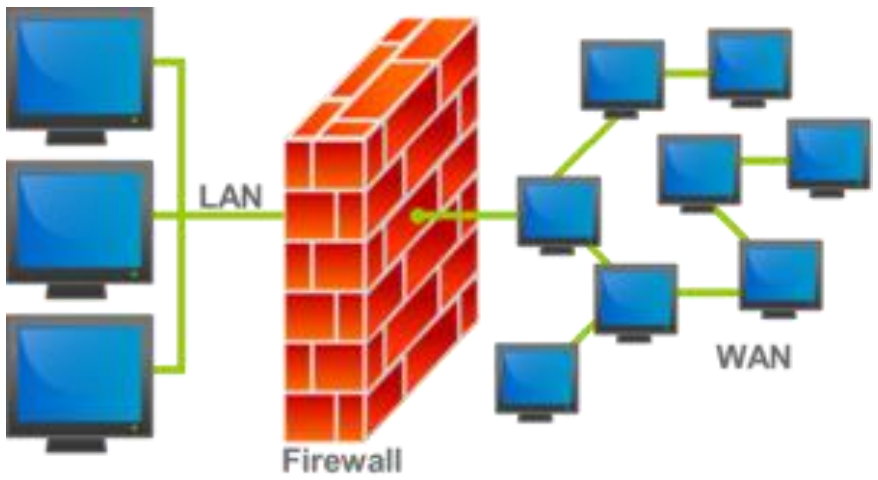
Requesting new email messages

192.168.23.100:143



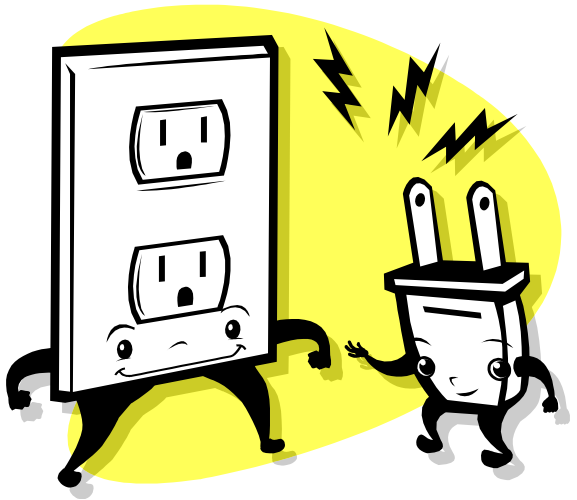
Firewalls

- Problem 4: You can't always get there from here:
  - Communication may by filtered by network
    - e.g. by a firewall at the border of Tech's network
    - e.g. by the wireless access point in Main Hall
  - Often by the port number



Sockets

- Socket API (Application Programming Interface)
  - Allows communication over IP (Internet Protocol)
  - Originally in Berkeley Unix
    - Thus: Berkeley sockets or BSD sockets
  - De facto standard in all operating systems
  - API in most programming languages:
    - C/C++
    - Java
    - C#
    - …



Java client: reading from a socket

- Step 1: Create a new Socket object
  - Needs to know IP address of server + port number

```
Socket socket = new Socket("127.0.0.1", 5000);
```

- Step 2: Create an InputStreamReader
  - Converts low-level socket data into characters

```
InputStreamReader stream = new InputStreamReader(socket.getInputStream());
```

- Step 3: Create a BufferedReader
  - Provides buffered reading of character stream

```
BufferedReader reader = new BufferedReader(stream);
```

- Step 4: Read some text

```
String message = reader.readLine();
```

BufferedReader

Method Summary	
void	close() Close the stream.
void	mark(int readAheadLimit) Mark the present position in the stream.
boolean	markSupported() Tell whether this stream supports the mark() operation, which it does.
int	read() Read a single character.
int	read(char[] cbuf, int off, int len) Read characters into a portion of an array.
String	readLine() Read a line of text.
boolean	ready() Tell whether this stream is ready to be read.
void	reset() Reset the stream to the most recent mark.
long	skip(long n) Skip characters.

Java client: writing to a socket

- Step 1: Create a new Socket object
  - Or use an existing one
  - You can combine reads and writes to same socket

```
Socket socket = new Socket("127.0.0.1", 5000);
```

- Step 2: Create an PrintWriter
  - Seen previously when writing to a file

```
PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
```

- Step 3: Write something

```
writer.println("Hello over there!");
```

PrintWriter

void	**print**(double d) Prints a double-precision floating-point number.
void	**print**(float f) Prints a floating-point number.
void	**print**(int i) Prints an integer.
void	**print**(long l) Prints a long integer.
void	**print**(Object obj) Prints an object.
void	**print**(String s) Prints a string.
PrintWriter	**printf**(Locale l, String format, Object... args) A convenience method to write a formatted string to this writer using the specified format string and arguments.
PrintWriter	**printf**(String format, Object... args) A convenience method to write a formatted string to this writer using the specified format string and arguments.
void	**println**() Terminates the current line by writing the line separator string.
void	**println**(boolean x) Prints a boolean value and then terminates the line.
void	**println**(char x) Prints a character and then terminates the line.
void	**println**(char[] x) Prints an array of characters and then terminates the line.

Just some of the methods in PrintWriter

Java socket server

- Client needs somebody to talk to!
- Server slightly different than client:
  - Must be running before client connects
  - Server decides port number to listen on
    - But doesn't specify IP address
    - Doesn't know who is going to connect
  - Blocks, waiting to *accept* an incoming client
  - Then reading/writing just as in client

Java socket server

- Step 1: Create a ServerSocket object
  - Declares what port you are listening on
  - Nobody else on the computer better be using it!

```
ServerSocket serverSock = new ServerSocket(5000);
```

- Step 2: Wait for a client to connect
  - accept() method blocks until client arrives
  - Returns a new Socket object for talking to client

```
Socket sock = serverSock.accept();
```

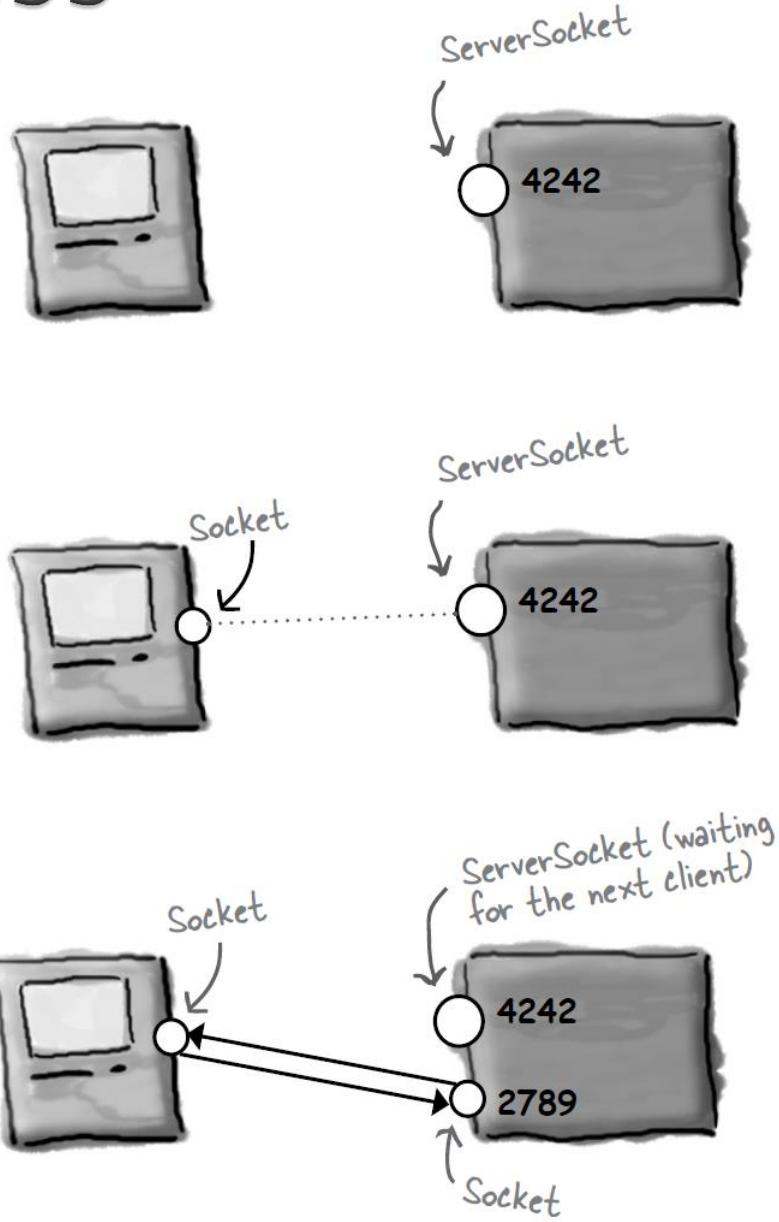
- Step 3: Read/write same way as a client
  - Create BufferedReader for reading strings
  - Create PrintWriter for writing strings

Connection process

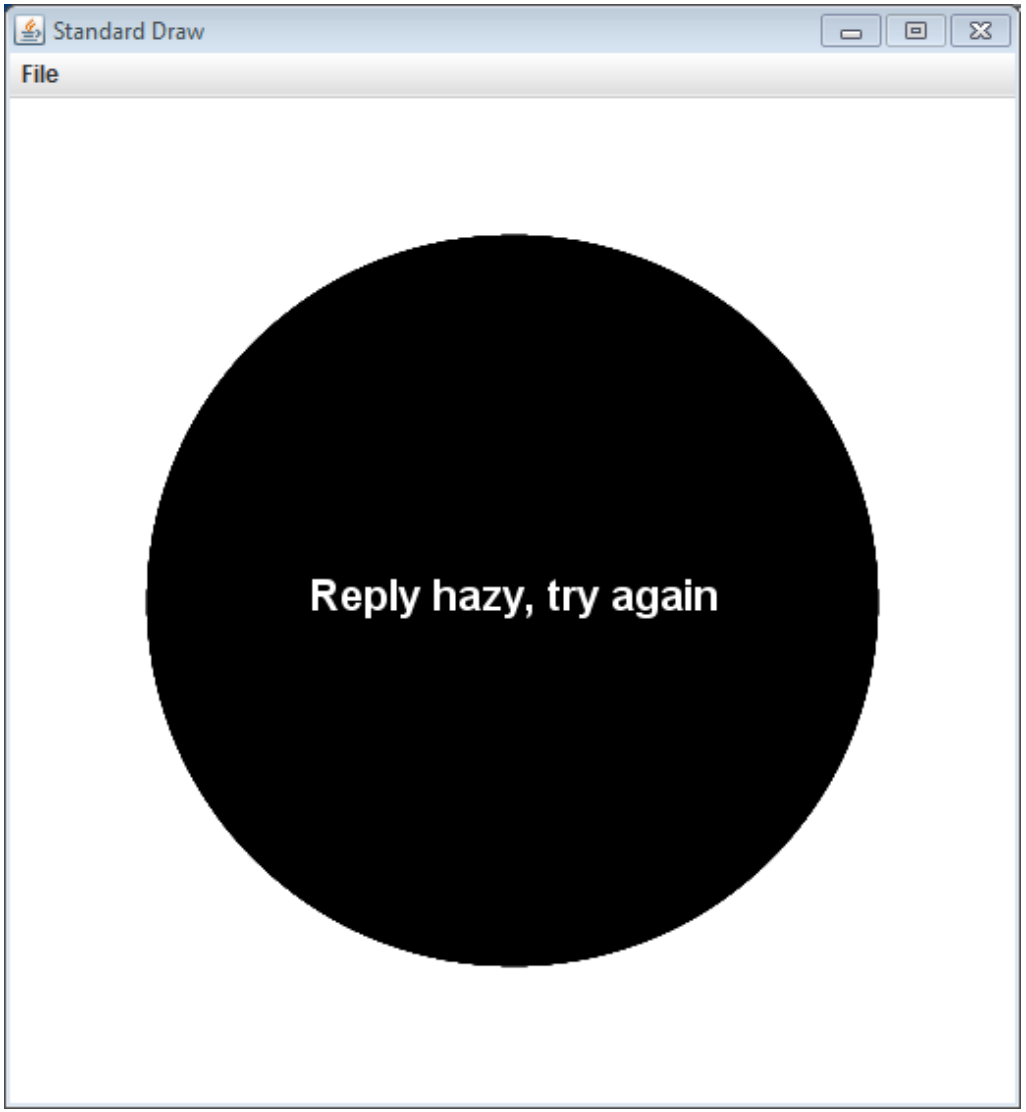
1. Server program starts up.
2. Starts listening on port 4242.
3. OS sends all inbound connection requests to 4242 to the server program.

4. Client program starts up
5. Requests connection to server IP address on port 4242.

6. Server establishes a socket connection to client, using outgoing port number 2789
7. Server can listen for new clients on the 4242 port number.



Socket clients and servers



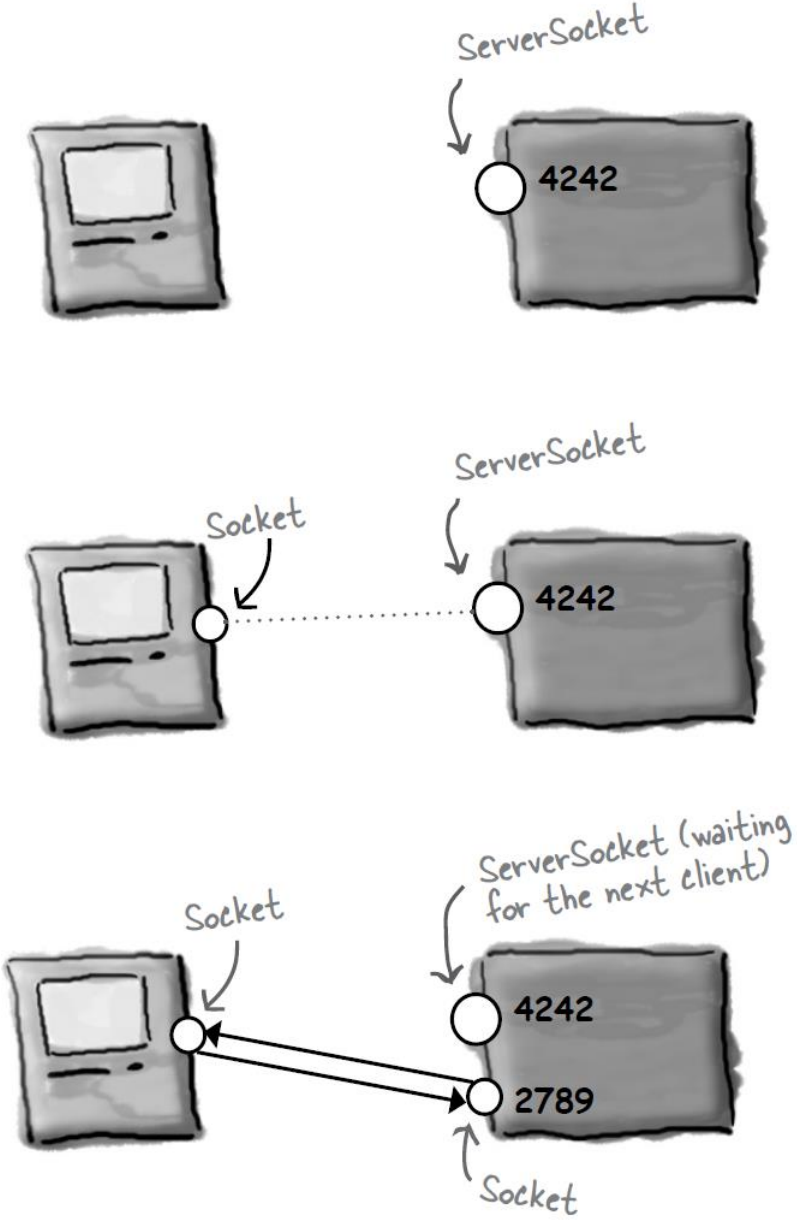
Magic 8 ball: Persistent connections

- Original version: One prediction per connection
- Persistent version:
  - A *protocol* between client and server

Client	Server
	Wait for client
Make connection to server	
Send name of user	
	Send first fortune
Receive first fortune	
Send "MORE"	
	Receive command "MORE"
	Send second fortune
Receive second fortune	
Send "QUIT"	
Close socket	Receive command "QUIT"
	Close socket

Magic 8 ball: Multi-threaded server

- Problem with persistent version:
  - One client can hog the 8-ball for a long time
- Multi-threaded server:
  - Spawn a thread to handle each client
  - Server's main thread can then wait for a new client



Summary

- Basics of networking
  - Computer all have a numeric IP address
    - Some computers have a friendly name (e.g. google.com)
  - Port numbers identify program to send request to
- Java socket communication
  - Clients create: `Socket` object
  - Servers create: `ServerSocket`, then a new `Socket` object per client that connects
  - Reading via `BufferedReader`
  - Writing via `PrintWriter`